REPAST SIMPHONY SYSTEM DYNAMICS GETTING STARTED

MARK BRAGEN

1. System Dynamics in Repast Simphony

New to this release of Repast Simphony is support for developing System Dynamics models from scratch. This release also supports importing existing System Dynamics models into your projects and converting it to a format that can be displayed and manipulated by the System Dynamics Graphical User Interface (GUI). Given the set of equations of the System Dynamics model, Java source code is generated that implements the model. The standard generated code implements the Euler method directly. An alternate form generated for System Dynamics models that meet certain criteria (specified in a later section), is compatible with the Apache ODE Solver library. This tutorial presents the steps required to create a System Dynamics model in Repast Simphony. Figure 1 shows the layout of the System Dynamics GUI screen in the Eclipse IDE. The layout is similar to all other Repast Simphony models that are specified via Eclipse. The left-side panel contains all the Repast Simphony projects that have been defined. Displayed in area are all files related to the projects such as source code, generated code, output files, etc. The top-center panel is a tabbed panel in which source code and diagrams can be viewed and edited. Finally, the lower-right panel is a tabbed panel in which console output, search output, System Dynamics model variable properties, etc. will be displayed. The Eclipse IDE is highly configurable and the more experienced users will know that location of these areas can easily be moved into other configurations to suit the needs of the user.

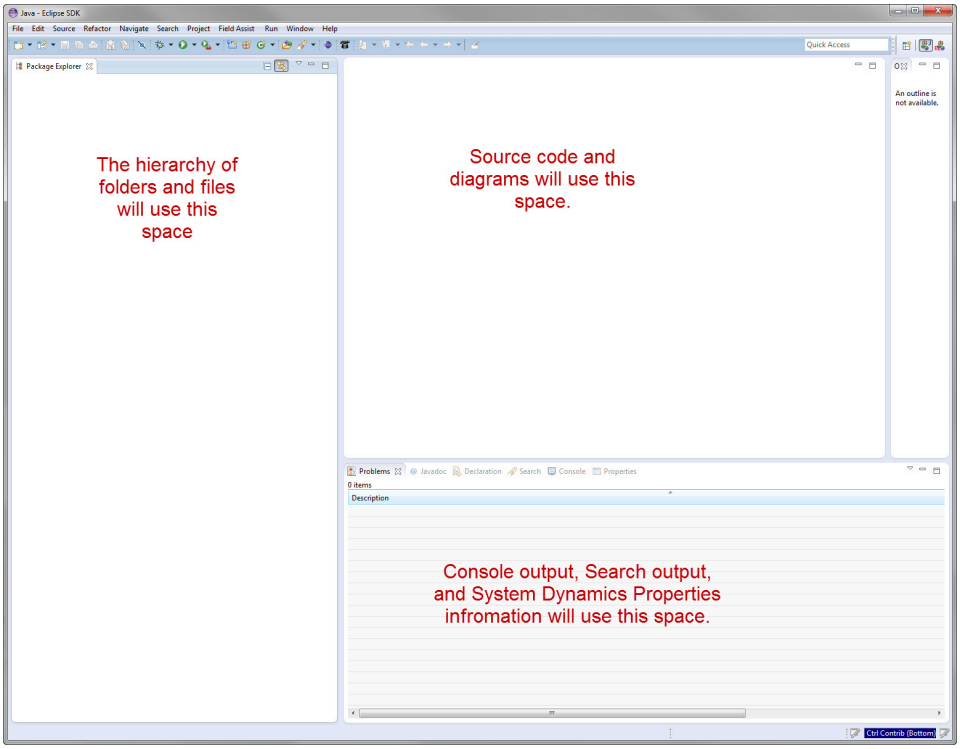

FIGURE 1

2. CREATE A NEW REPAST SIMPHONY PROJECT

The first step in creating a System Dynamics (SD) model is to create a Repast Simphony project in which the SD model will be located. Figure 2 illustrates how this is done. Right click the mouse in the Project panel to bring up the menus. Navigate from New to Other and left click on Other. Another window will open that lists the available wizards for performing tasks. This is shown in figure 3. Select Repast Simphony Project and left click on the Next button. Another window appears in which you specify the name of the project. See Figure 4. Note that the project name must be unique within your workspace. Left click on Finish to cause the new project wizard to create the required Repast Simphony project structure.

When the create Repast Simphony wizard completes, you will see the scenario directory structure as contained in Figure 5. The wizard automatically created two files under source: ModelInitializer.groovy and ModelInitializer.agent. These files are used by other Repast Simphony modeling techniques and are not required for System Dynamics models. You can delete them from your. Leaving them in the project will not affect your model.

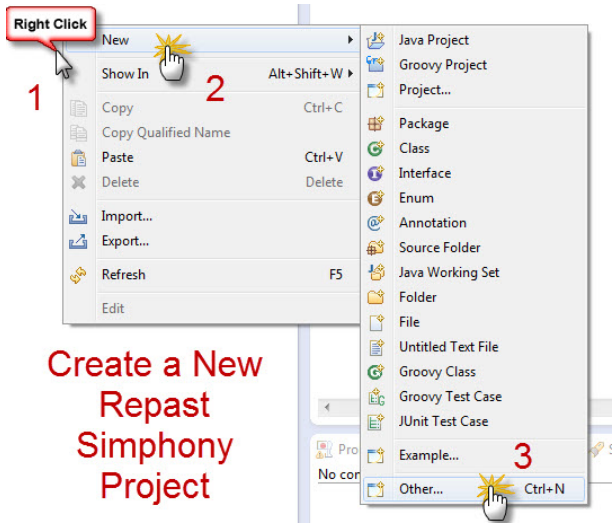

FIGURE 2

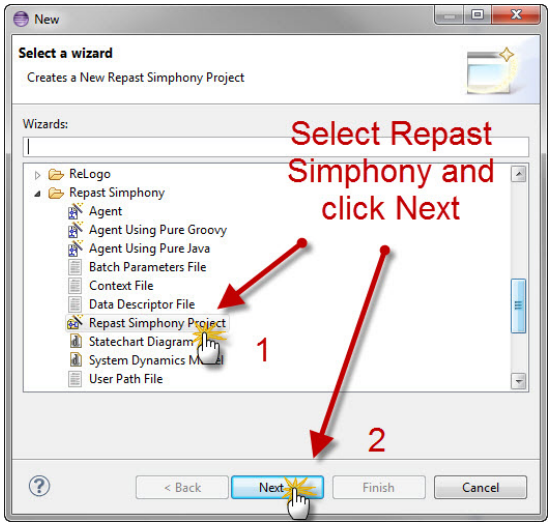

FIGURE 3

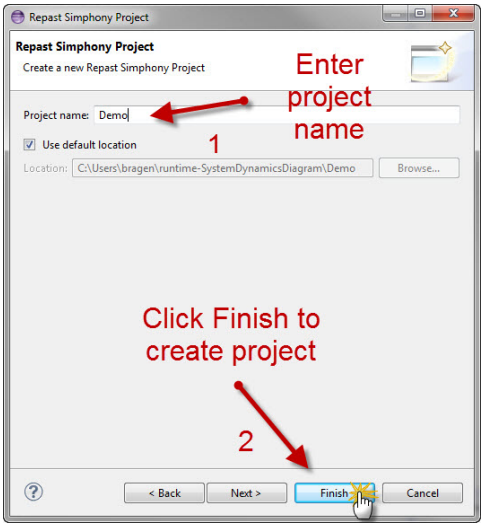

FIGURE 4

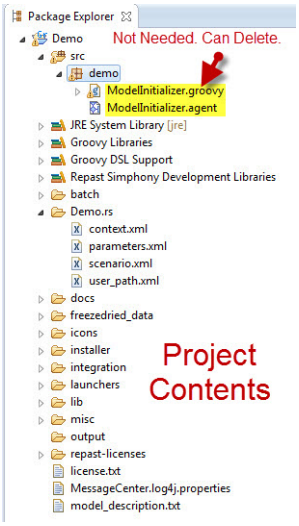

FIGURE 5

3. Create an Empty System Dynamics Model

Now that we have the project created, we need to create an empty System Dynamics model to initialize the dataset in which the model will be stored. Once again, theres a wizard for that. Right click on the package name demo under the src folder. Navigate to New and Other. Click on Other (See Figure 6). The list of available wizards will appear as it did when the project was being created. Select System Dynamics Model and click on Next (See Figure 7).

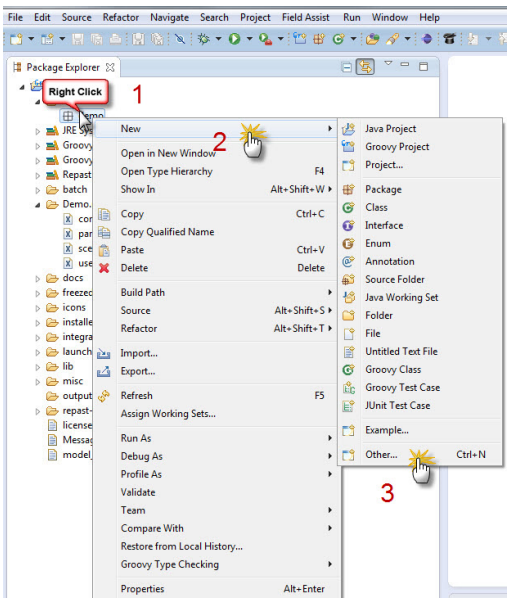

Figure 6

A new window will appear requiring four pieces of information (See Figure 8):

(1) The folder into which to place the model file. This is initialized to the folder that was selected when the New menu was selected.
(2) The name of the model file (.rsd extension). This is initialized to default.rsd.
(3) The class name. This is the name of the Java class that will be generated. It is initially blank and must be supplied.
(4) The package name. This is the Java package name into which the class will be placed. It is initially blank and must be supplied.

Click on Finish to complete the process of creating a new SD model.

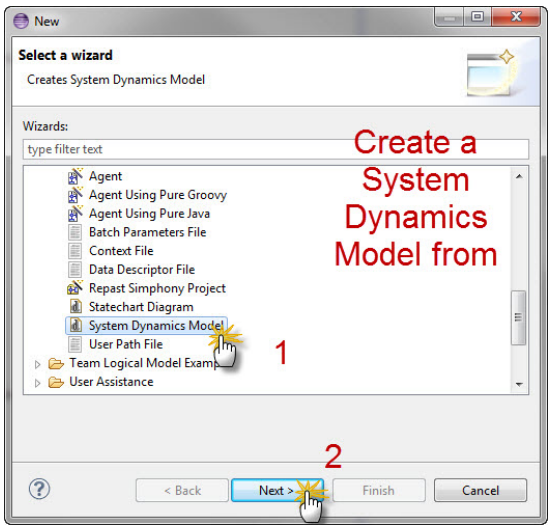

FIGURE 7

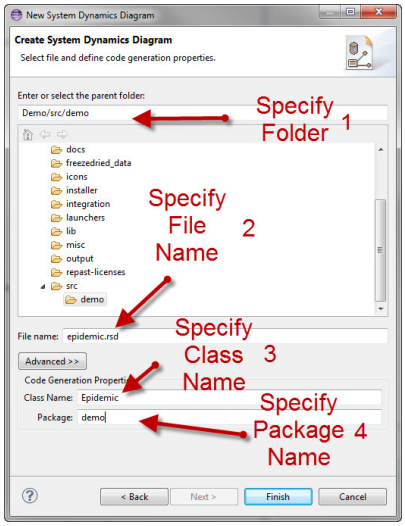

FIGURE 8

4. Start Building the Model

When the wizard completes the process of creating an empty SD model, the diagram tab for the model will appear (See Figure 9). On the right side of the panel, there is a set of SD objects that can be used in the SD model. There are: Influence Arrows, Rates, Stocks, Constants, Variables, Clouds, and Lookups.

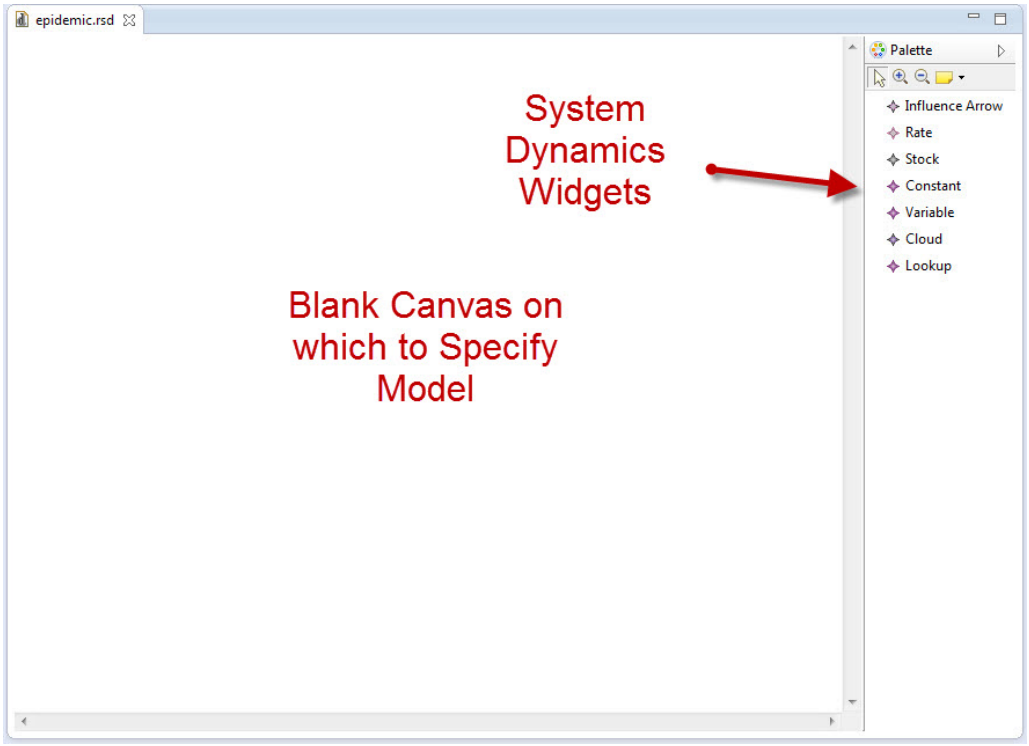

Figure 9

5. Add a stock

Begin the process of building the SD model by clicking on the Stock object in the Palette and then click anywhere in the diagram panel to place the stock into it. You will be placed into the mode in which you specify the variable name for the stock. Enter Healthy and hit enter. See Figures 10, 11, and 12.

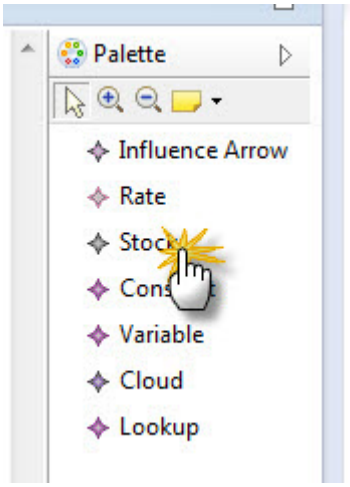

FIGURE 10

FIGURE 11

FIGURE 12

6. Variable Properties

Select the Properties tab in the bottom panel of the Eclipse IDE (See Figure 13). The required data for the stock variable are:

(1) Name (set to name you specified).
(2) Type (set to stock since variable was declared as stock).
(3) Units for the Left Hand Side (LHS) of equation.
(4) LHS of equation (Initially set to Name you might change this depending on the structure of your model e.g. using arrays)
(5) The single argument for the INTEG function. Note that any legal expression that results in one double value is allowable.
(6) An Initial Value

Comments are optional, though recommended. Functions and Influencing variables are for convenience. Since no influence arrows have been included in the model as of yet, this list box is empty.

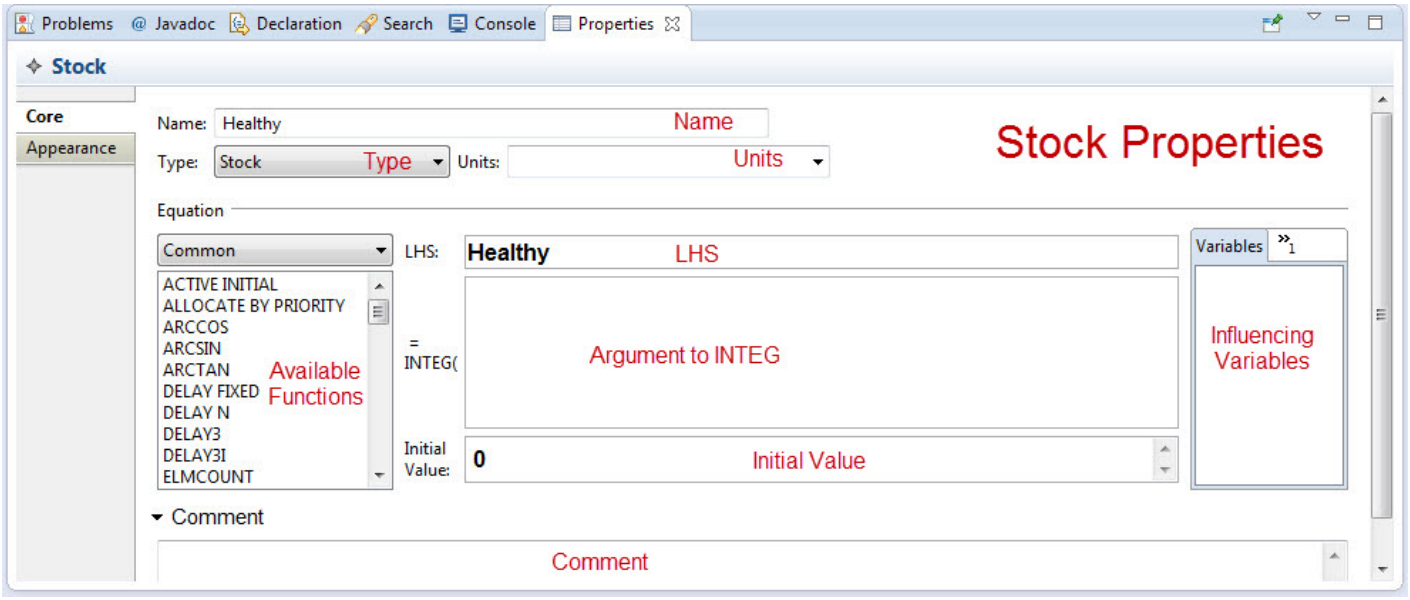

Figure 13

7. Completing the Diagram

At this point, create additional model structure Click on object in pallete and then click on diagram to insert it. To create an arrow or rate, click on pallete object, click on the arrow source and hold the mouse button down while moving the cursor to the destination variable and release the mouse button. Rates require a name. Influence arrows do not.

(1) Create a constant named initial susceptible
(2) Create another stock named Infected
(3) Create an influence arrow from initial susceptible to Health
(4) Create a rate arrow between Healthy and Infected

You should see something like Figure 14.

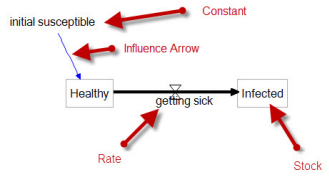

Figure 14

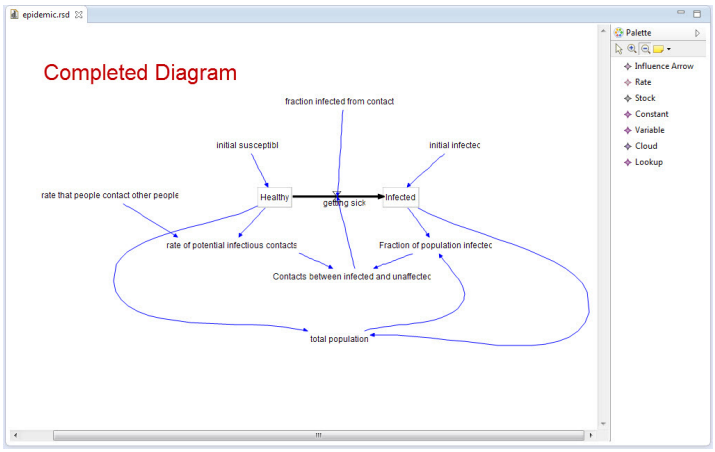

Figure 15

8. System Level Model Parameters

Each SD model has a set of required parameters. They are the model start and end times, the integration interval, the reporting interval and finally the unit of time. These are specified on the System Model properties tab. This tab becomes available by clicking in any empty area of the SD model diagram. See Figure 16.

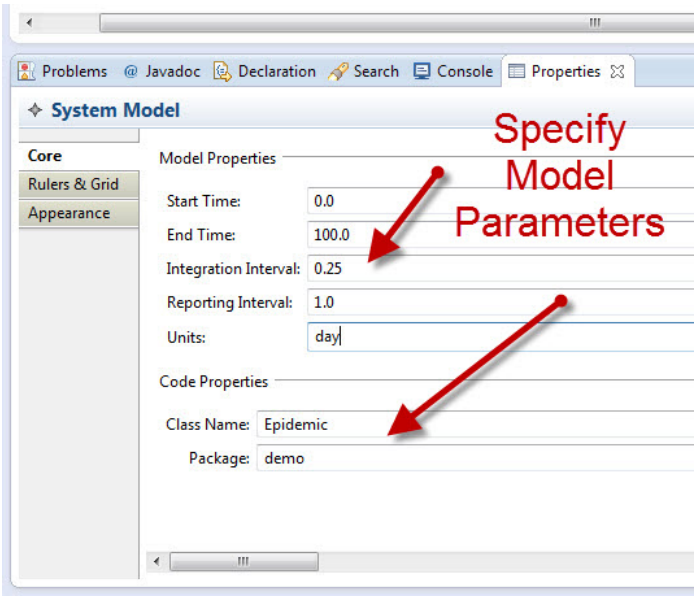

Figure 16

9. Specifying Variable Properties

Now that the model structure is complete, it is time to specify the data for all the variables. Start with the Healthy stock (Figure 17).

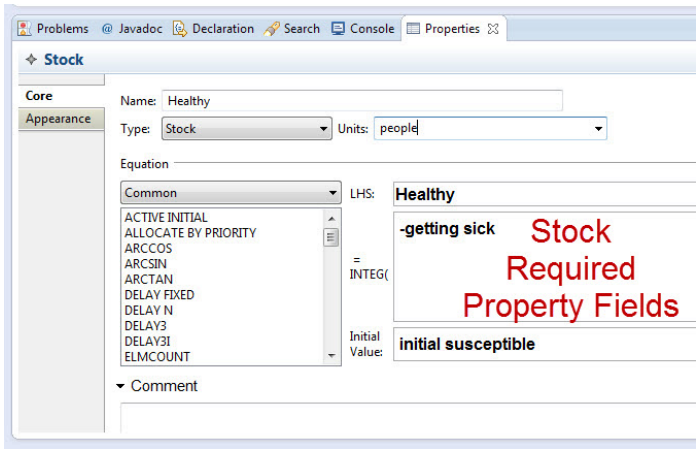

Figure 17

Then getting sick (Figure 18).

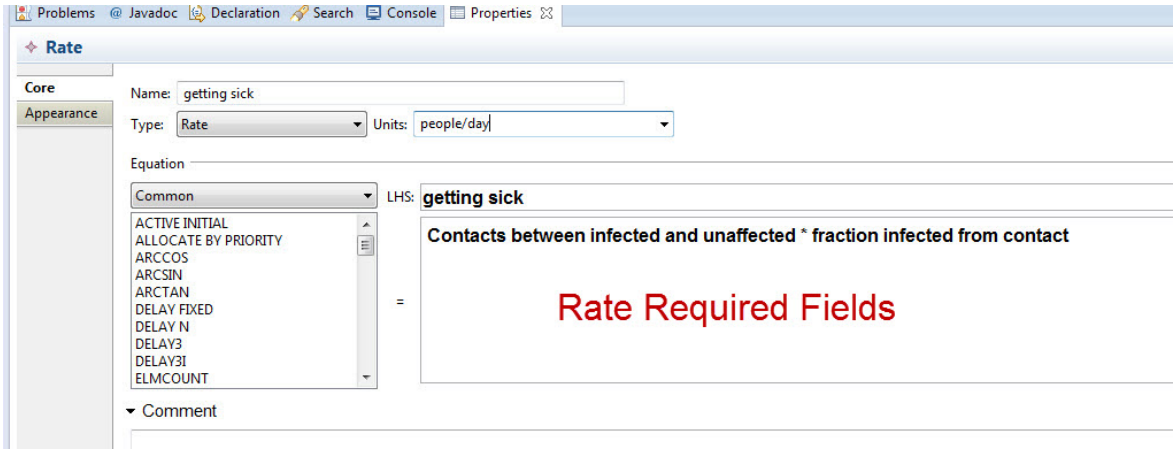

Figure 18

Then fraction infected from contact (Figure 19).

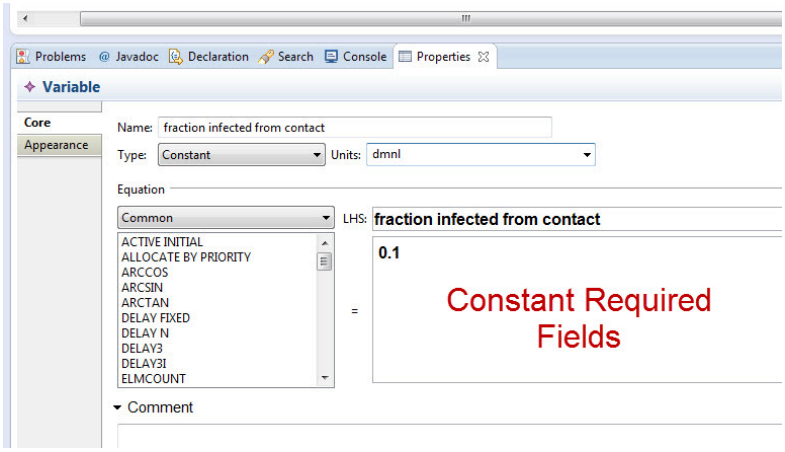

FIGURE 19

Then contacts between infected and unaffected (Figure 20). To illustrate the consistency check, leave the units field blank for now.

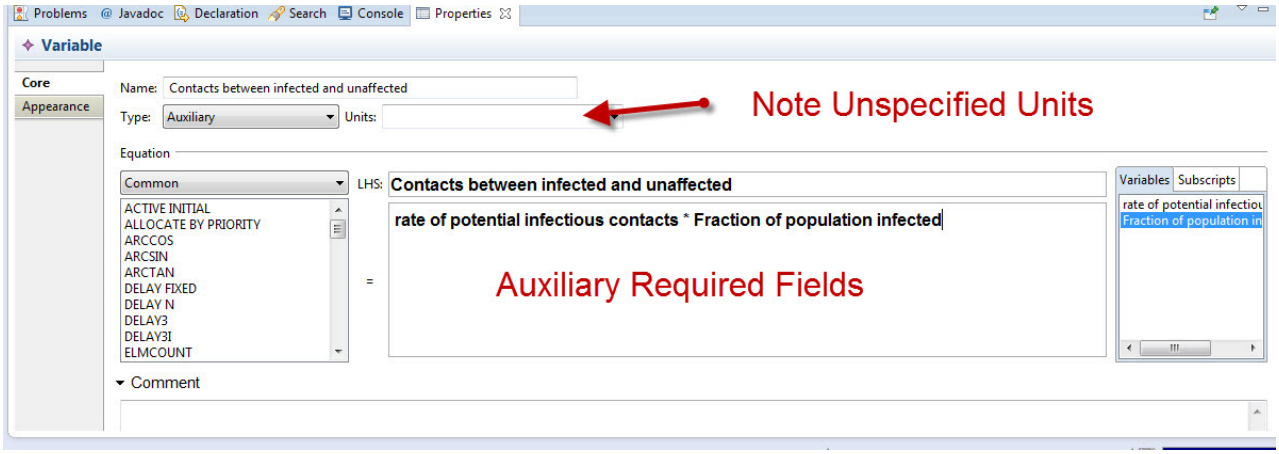

FIGURE 20

10. All Model Equations

Complete the properties for all the variables in the model. Here is a complete specification of the equations and units.

```
rate that people contact other people= 5
1/day

Contacts between infected and unaffected =
rate of potential infectious contacts  * Fraction of population infected
people/day

Infected  = INTEG (getting sick, initial infected)
people

total population = Healthy + Infected
people

fraction infected from contact = 0.1
dmnl

Fraction of population infected = Infected / total population
dmnl

getting sick = Contacts between infected and unaffected  *
                      fraction infected from contact
people /day

initial infected = 10
people

initial susceptible = 1e+006
people

rate of potential infectious contacts = Healthy
                   * rate that people contact other people
people /day

Healthy = INTEG( - getting sick, initial susceptible)
people
```

11. Checking Model Consistency

Check the consistency of the SD model (Figure 21).

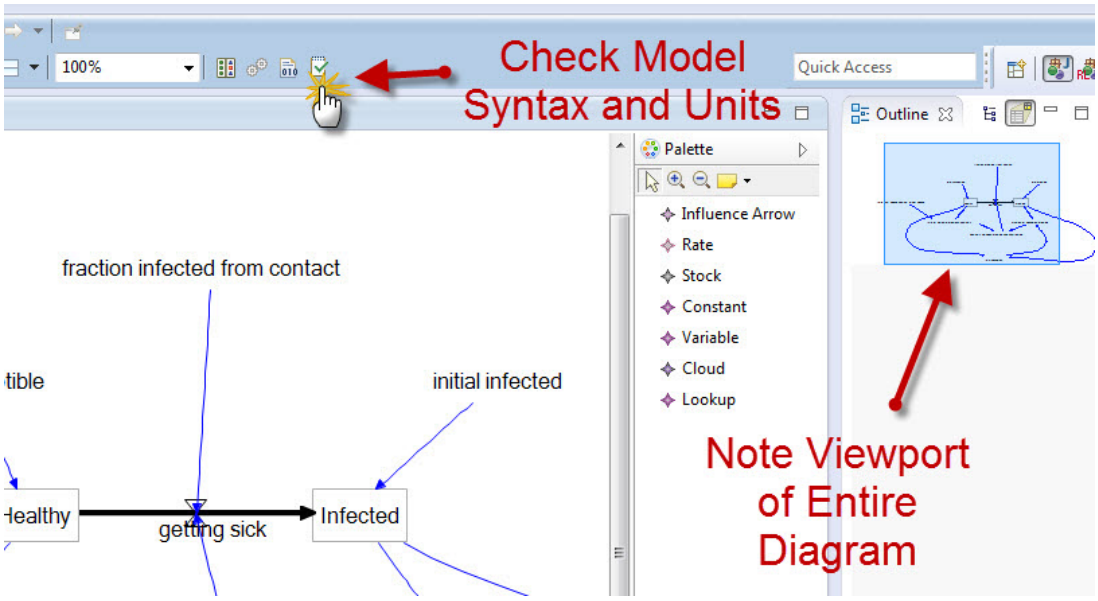

Figure 21

There are inconsistencies in the model. Specifically, Contacts between infected and unaffected does not have its units specified. (Figure 22). Set the Units to people/day (Figure 23).

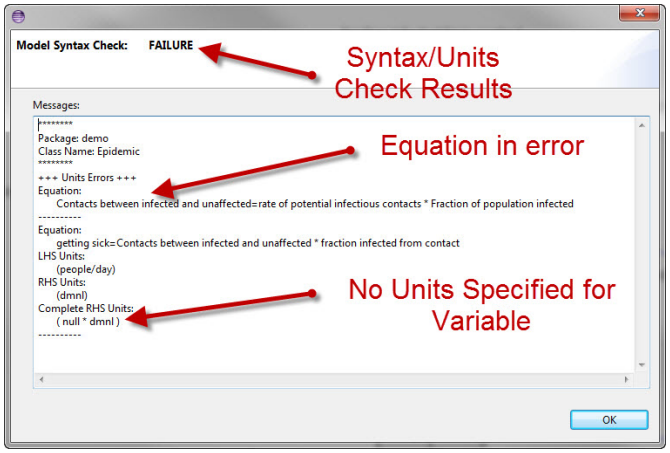

FIGURE 22

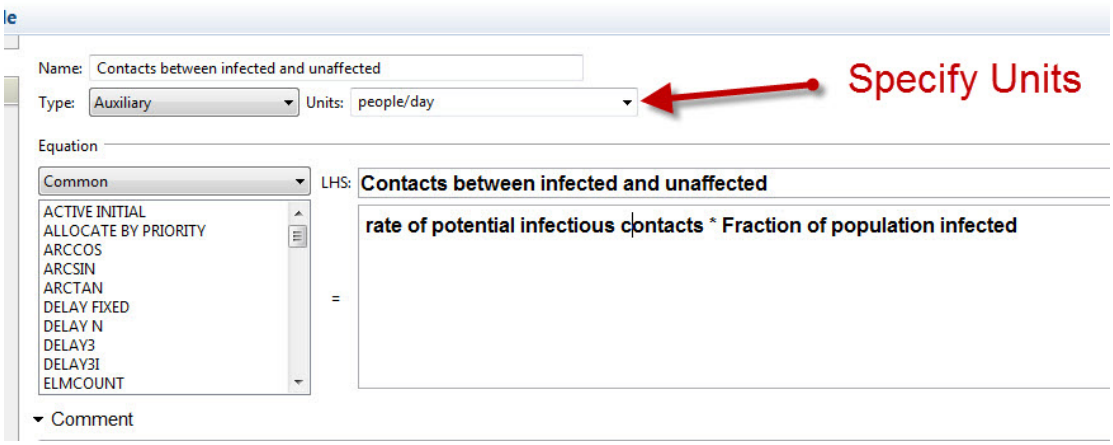

FIGURE 23

Check for model consistency once again. The model is found to be consistent. (Figure 24).

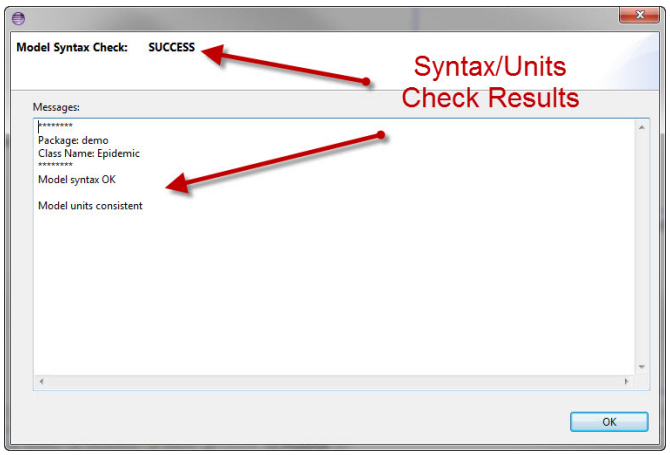

FIGURE 24

12. GENERATE THE JAVA CODE

Now generate the code (See Figure 21, icon to the left of check consistency). Files other than source code will be generated in the scenario directory by Repast Simphony. Among these are:

(1) context.xml
(2) parameters.xml
(3) repast.simphony.action.data_set_0.xml
(4) repast.simphony.dataLoader.engine.ClassNameDataLoaderAction_1.xml
(5) scenario.xml
(6) usath.xm

Other files can be created in this directory when you create objects such as graphs and output files through the Repast Simphony GUI when you run the model. The user is prompted whether ir not the scenario directory will be reinitialized to its default contents. (Figure 25).

The Java source code and scenario files have been successfully been generated (Figure 26). Figure 27 shows the generated Java source code and scenario files.

Figure 28 contains the declaration of the model variables and sample getters. Note that the variable names you specified in the equations have been altered to be legal Java variable names.

Any variable that has a numeric constant as its value will be placed into the parameters.xml file so that these values can be set via the RS GUI rather than having to edit

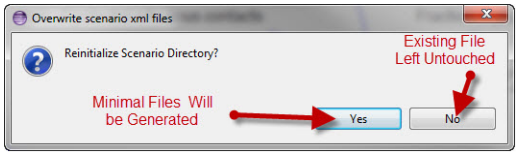

FIGURE 25

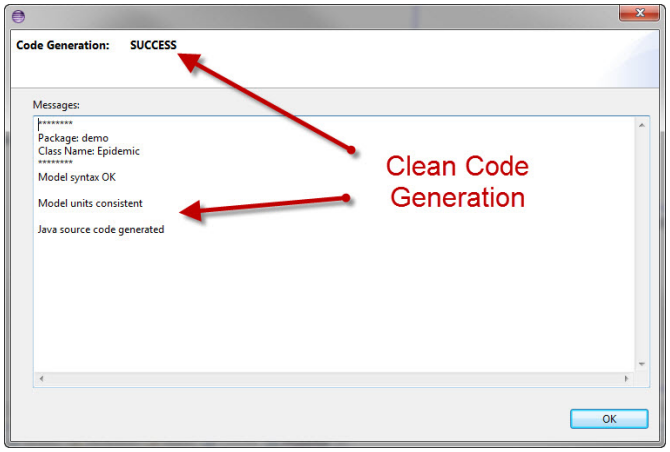

FIGURE 26

the diagram and regenerate the code. Figure 29 contains a code sample for extracting this data from the parameters.xml file.

Figure 30 shows a code sample for the implementation of a stock variable.

Figure 31 contains the source code for the ContextBuilder.

Figure 32 shows the parameters.xnl file that was generated. Note that the values specified in the SD model diagram have been incorporated in to the parameters file.

Figure 33 contains a portion of the automatically generated data set for the variables that are used in the model. This is for the convenience of the user in creating reports and graphs.

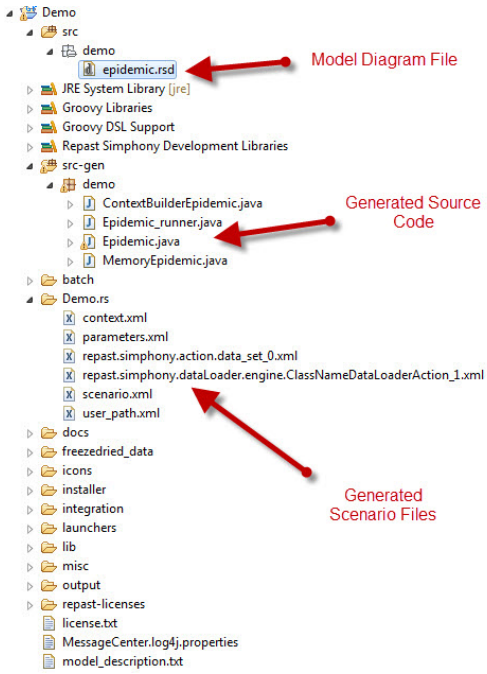

FIGURE 27

```
public class MemoryEpidemic {
public double Contacts_between_infected_and_unaffected;
public double FINAL_TIME;
public double Fraction_of_population_infected;
public double Healthy;
public double INITIAL_TIME;
public double Infected;
public double NAREPLACEMENT;
public double SAVEPER;
public double TIME_STEP;
public double Time;
public double fraction_infected_from_contact;
public double getting_sick;
public double initial_infected;
public double initial_susceptible;
public double rate_of_potential_infectious_contacts;
public double rate_that_people_contact_other_people;
public double total_population;
public MemoryEpidemic() {
}

public double get_Contacts_between_infected_and_unaffected() {
return Contacts_between_infected_and_unaffected;
}
public double get_FINAL_TIME() {
return FINAL_TIME;
```

Declaration of model variables

FIGURE 28

```
/*
 * Equation: rate that people contact other people=5
 *
 * Units:1/day
 *
 * Comment: None Provided
 */
{
    memory.rate_that_people_contact_other_people = (Double) params
            .getValue("rate_that_people_contact_other_people"); // 2;
    logit("memory.rate_that_people_contact_other_people",
            getINITIALTIME(),
            (Double) params
                    .getValue("rate_that_people_contact_other_people"),
            memory.get_SAVEPER());
}
```

Retrieve Value From Parameters File

FIGURE 29

```
protected void repeated0(double time, double timeStep) {

    {
        /*
         * Equation: Healthy= INTEG(-getting sick,initial susceptible)
         *
         * Units:people
         *
         * Comment: None Provided
         */
        double _t0 = 0.0;
        _t0 = sdFunctions.INTEG("Healthy", memory.Healthy, time, timeStep,
                (-(memory.getting_sick)), (memory.initial_susceptible));
        memory.Healthy = _t0;
        logit("memory.Healthy", time, _t0, memory.get_SAVEPER());
    }
```

Stock Implementation

FIGURE 30

ContextBuilderEpidemic.java

```
package demo;

import repast.simphony.context.Context;
public class ContextBuilderEpidemic implements ContextBuilder<Object> {
    @Override
    public Context<Object> build(Context<Object> context) {
        Epidemic on = new Epidemic("Epidemic");
        on.oneTime();
        context.setId("Epidemic");
        context.add(on);
        context.add(on.getMemory());
        RunEnvironment.getInstance().endAt(
                (on.getMemory().getFINALTIME() - on.getMemory()
                        .getINITIALTIME()) / on.getMemory().getTIMESTEP());
        return context;
    }
}
```

Generate Objects and Schedule End Time

FIGURE 31

```
<?xml version="1.0" encoding="UTF-8" ?>
<parameters>
<parameter name="randomSeed" displayName="Default Random Seed" type="int"
    defaultValue="__NULL__"
    isReadOnly="false"
    converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
/>
<parameter name="fraction_infected_from_contact" displayName="fraction infected from contact" type="double"
    defaultValue="0.1"
    isReadOnly="false"
    converter="repast.simphony.parameter.StringConverterFactory$DoubleConverter"
/>
<parameter name="initial_susceptible" displayName="initial susceptible" type="double"
    defaultValue="1000000.0"
    isReadOnly="false"
    converter="repast.simphony.parameter.StringConverterFactory$DoubleConverter"
/>
<parameter name="TIME_STEP" displayName="TIME STEP" type="double"
    defaultValue="0.25"
    isReadOnly="false"
    converter="repast.simphony.parameter.StringConverterFactory$DoubleConverter"
/>
<parameter name="initial_infected" displayName="initial infected" type="double"
    defaultValue="10.0"
    isReadOnly="false"
    converter="repast.simphony.parameter.StringConverterFactory$DoubleConverter"
/>
<parameter name="rate_that_people_contact_other_people" displayName="rate that people contact other people" type=
    defaultValue="5.0"
    isReadOnly="false"
    converter="repast.simphony.parameter.StringConverterFactory$DoubleConverter"
/>
<parameter name="INITIAL_TIME" displayName="INITIAL TIME" type="double"
    defaultValue="0.0"
    isReadOnly="false"
    converter="repast.simphony.parameter.StringConverterFactory$DoubleConverter"
/>
```

Generated Parameters.xml

FIGURE 32

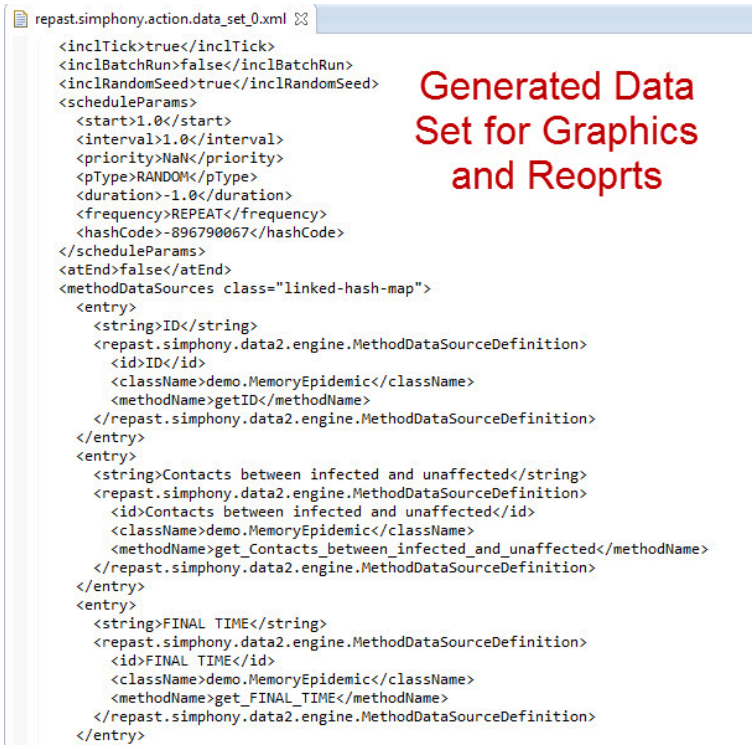
repast.simphony.action.data_set_0.xml
Generated Data Set for Graphics and Reoprts
<inclTick>true</inclTick>
<inclBatchRun>false</inclBatchRun>
<inclRandomSeed>true</inclRandomSeed>
<scheduleParams>
<start>1.0</start>
<interval>1.0</interval>
<priority>NaN</priority>
<pType>RANDOM</pType>
<duration>-1.0</duration>
<frequency>REPEAT</frequency>
<hashCode>-896790067</hashCode>
</scheduleParams>
<atEnd>false</atEnd>
<methodDataSources class="linked-hash-map">
<entry>
<string>ID</string>
<repast.simphony.data2.engine.MethodDataSourceDefinition>
<id>ID</id>
<className>demo.MemoryEpidemic</className>
<methodName>getID</methodName>
</repast.simphony.data2.engine.MethodDataSourceDefinition>
</entry>
<entry>
<string>Contacts between infected and unaffected</string>
<repast.simphony.data2.engine.MethodDataSourceDefinition>
<id>Contacts between infected and unaffected</id>
<className>demo.MemoryEpidemic</className>
<methodName>get_Contacts_between_infected_and_unaffected</methodName>
</repast.simphony.data2.engine.MethodDataSourceDefinition>
</entry>
<entry>
<string>FINAL TIME</string>
<repast.simphony.data2.engine.MethodDataSourceDefinition>
<id>FINAL TIME</id>
<className>demo.MemoryEpidemic</className>
<methodName>get_FINAL_TIME</methodName>
</repast.simphony.data2.engine.MethodDataSourceDefinition>
</entry>

FIGURE 33

13. Execute the Model

At this point, the model is ready to be executed. Figure 34 shows the procedure for executing the model using the automatically generated launcher (available through the run¿configurations menu item or via the run configurations shortcut).

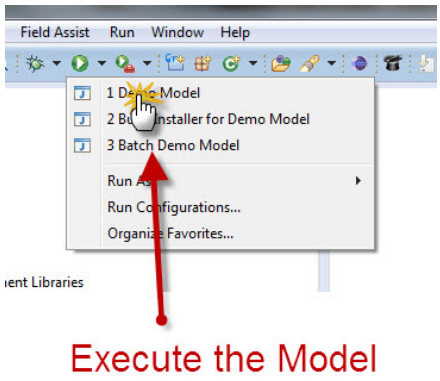

Figure 34

Figure 35 shows the Repast Simphony GUI at startup time. In the scenario tree panel, you can see the generated ContextBuilder and the predefined data set.

Lets first create a time series graph so that we can view the results of the model. Figures 36-39 illustrate the process of defining the graph.

Specify any name that you would like and use the predefined data set.

Select Healthy, Infected, and getting sick from the left hand side of the shuttle and transfer them to the right hand side. Click Next to continue.

Specify the Title and Y Axis label. Click finish to complete the graph definition. Remember to save the scenario. If you do not save the scenario, this information will be lost if you either exit Repast Simphony or reinitialize the model.

Figure 40 shows the process for initializing the model. This consists of access the scenario files and initializing all the variables that can be initialized at this time. In addition, the graph we defined will be displayed (Figure 41). Note that you can start the model without initializing it. The initialization process will take place, but this can happen so quickly that the GUI will not be able to display a graph before the model reaches its end time. It is recommended that you always initialize the model before starting it.

Figure 41 shows the GUI after the model has been initialized.

Figure 42 shows the icon to click to start the model execution.

Figure 43 shows the graph being displayed at the end of the model time frame. Note that the graph is updated as the model executes rather than just displaying at the end.

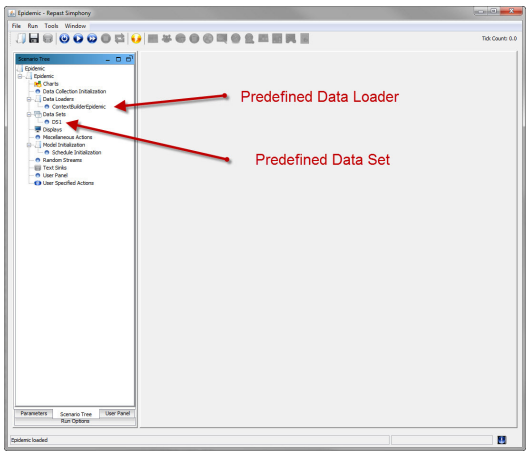

FIGURE 35

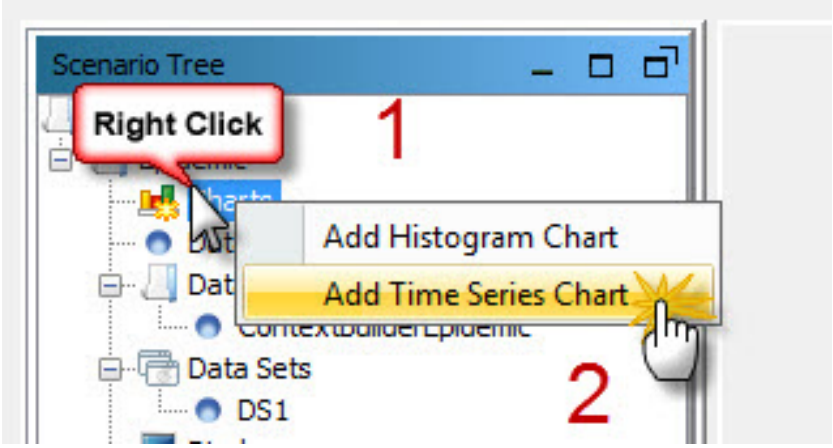

FIGURE 36

Figure 44 shows the icon that can be used to reset the model for another execution without having to restart the Repast Simphony GUI.

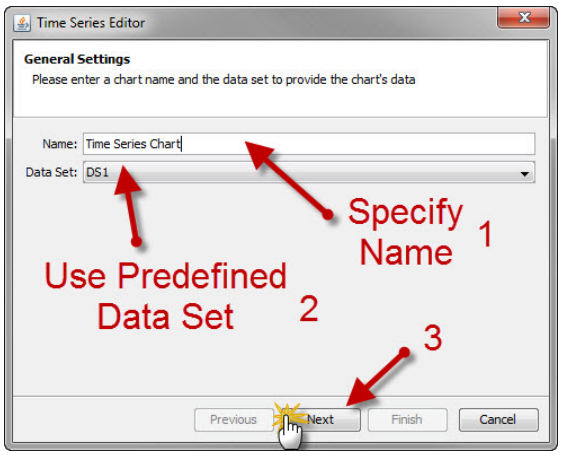

FIGURE 37

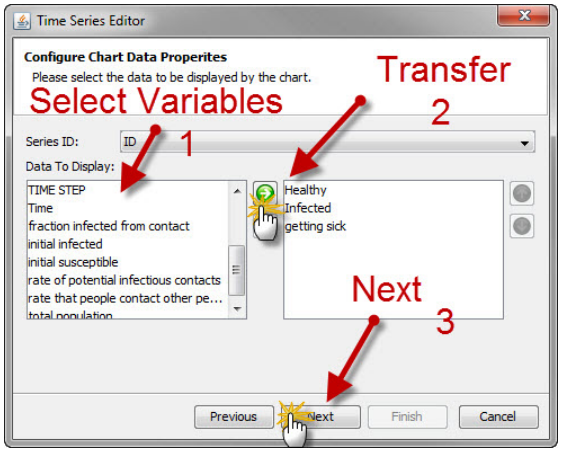

FIGURE 38

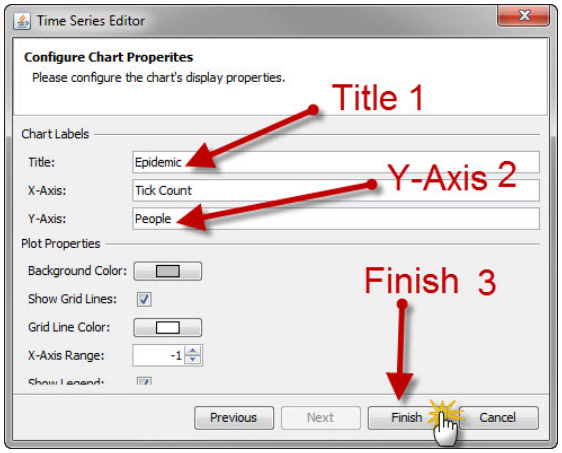

FIGURE 39

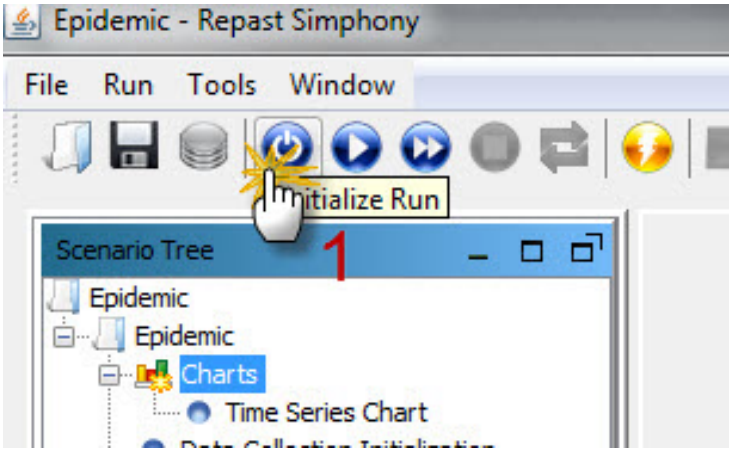

FIGURE 40

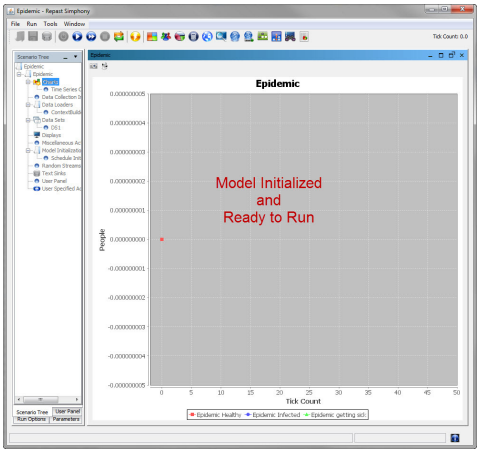

FIGURE 41

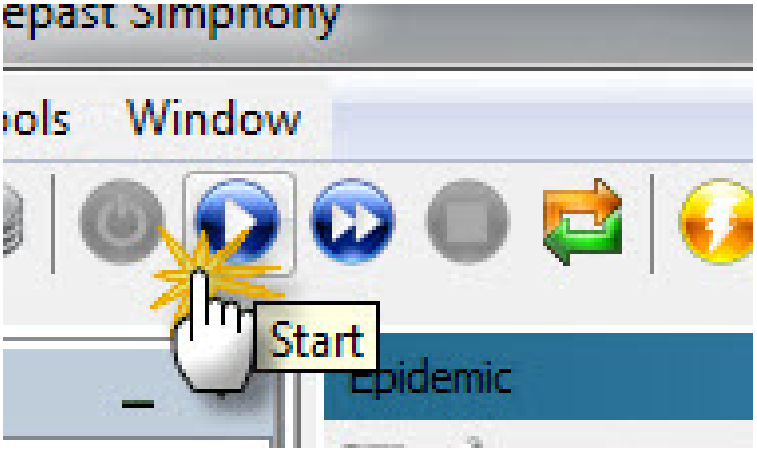

FIGURE 42

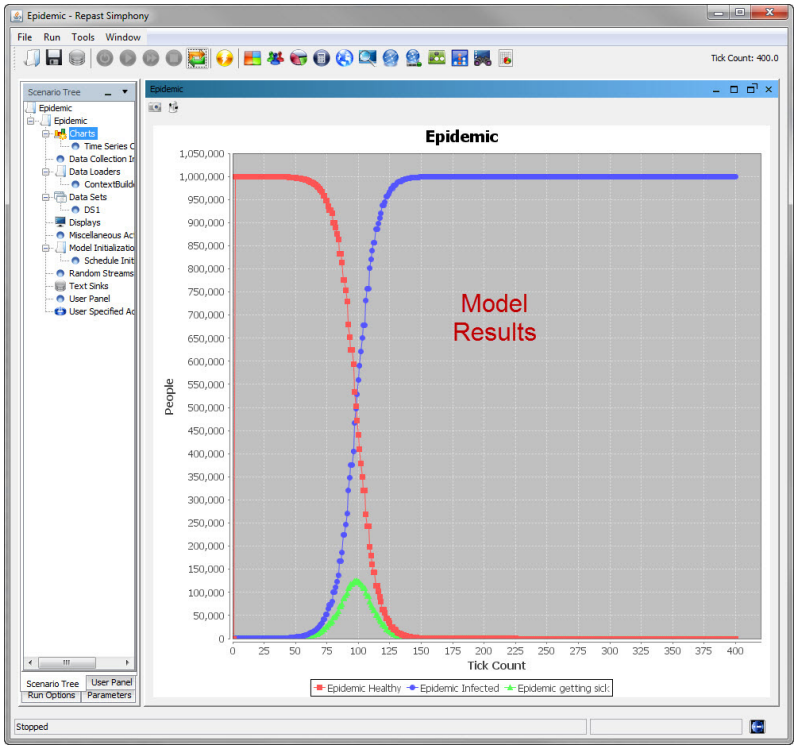

FIGURE 43

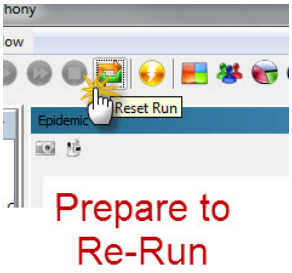

FIGURE 44

14. Import an Existing System Dynamics Model

The Repast Simphony System Dynamics subsystem can read the equations and associated data from mdl files generated by Vensim. This allows Repast Simphony to import existing System Dynamics models into your project. Figure 45 and 46 show the process of importing an existing system dynamics model into Repast Simphony. Note that a file browser pops up to allow the user to select the appropriate file.

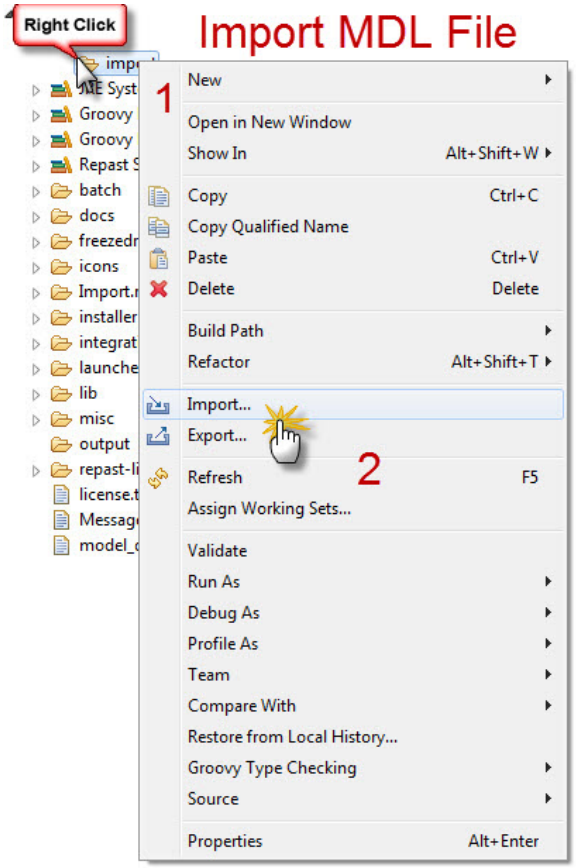

Figure 45

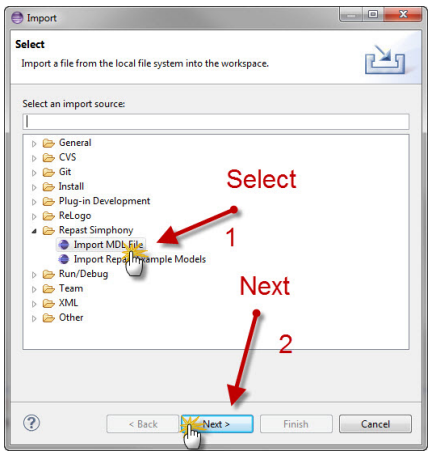

FIGURE 46

Once imported, the model diagram is displayed as in Figure 47. Note that positional data in the existing SD model is not transferred to Repast Simphony. Rather, the Eclipse diagramming software computes the initial positioning. Components can be rearranged by dragging them to the desired locations. Occasionally, influence arrows will not properly display (i.e. are not visible in the diagram). Minor shifting of the variables position normally causes the arrow to be displayed properly on the redraw.

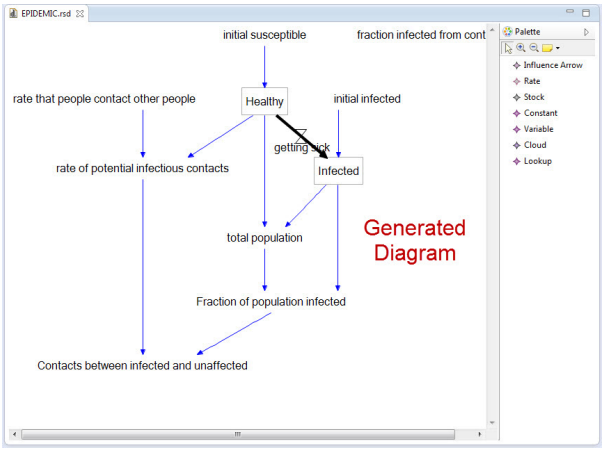

FIGURE 47

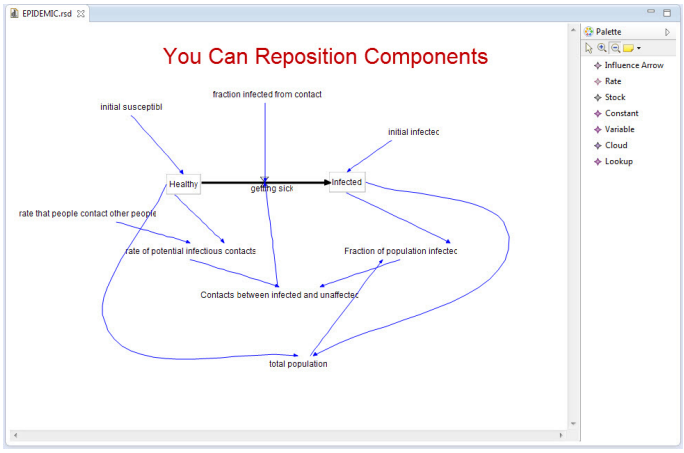

FIGURE 48

15. SUPPORT FOR ARRAYS

Repast Simphony System Dynamics supports the use of arrays in models. Figures 49-55 show the process for adding arrays to an existing model. First, we need to start the process by creating a subscript (Figure 49).

You must click on the Add button to bring up an empty, unnamed subscript (Figure 50).

Then we can add the name and specify the subscript values as a comma separated list (Figures 51 and 52). Note that subscripts take alphanumeric values. During the code generation phase, these are changed to numeric subscripts. It is intended that this be hidden from the user as the use of mapped subscripts (an advanced SD model technique) can cause the numeric mapping to be less that obvious in some cases.

The user needs to specify exactly where each instance of a subscript must appear. Not all variables in an equation that contains arrays need to be subscripted. But once a variable is referenced as an array, it must be referenced as an array in all instances with consistent use of subscripts. Figures 53 and 54 show the process of adding a subscript to an equation. In this case, we want to add the Country subscript to the Healthy stock variable to create an array. Place the cursor in the proper position within the LHS text box and then click on the subscript you wish to insert. The subscript notation is added to the Healthy stock variable.

As indicated in Figure 55, the subscript notation will be inserting only at the cursor location. If there are multiple variables requiring the subscript notation, each must be added individually.

Figure 56 shows on of the possible errors resulting from inconsistent array and non-array references to a variable. The instance of the Healthy stock variable in the rate of

FIGURE 49

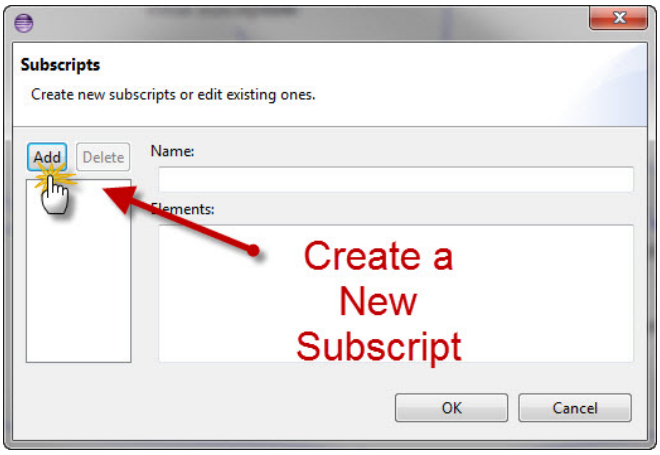

FIGURE 50

potential infectious contacts equation is that of a scalar reference. The consistency checker attempted to find the variable named Healthy in the scalar variable table. But since it had been defined as an array in its LHS definition, the lookup failed and the error is reported. This also occurs for the Infected and total population variables in this example.

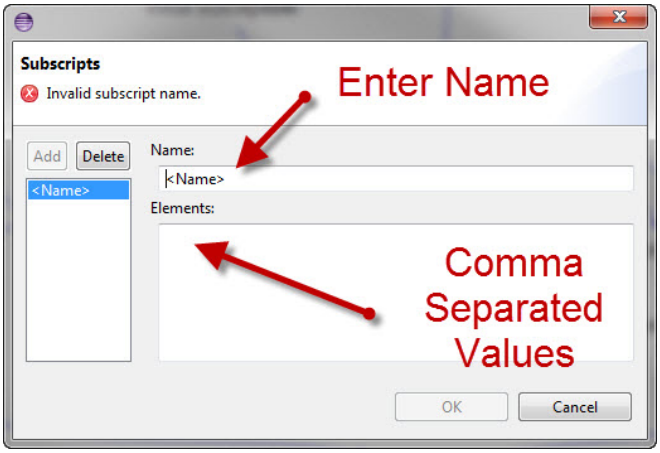

FIGURE 51

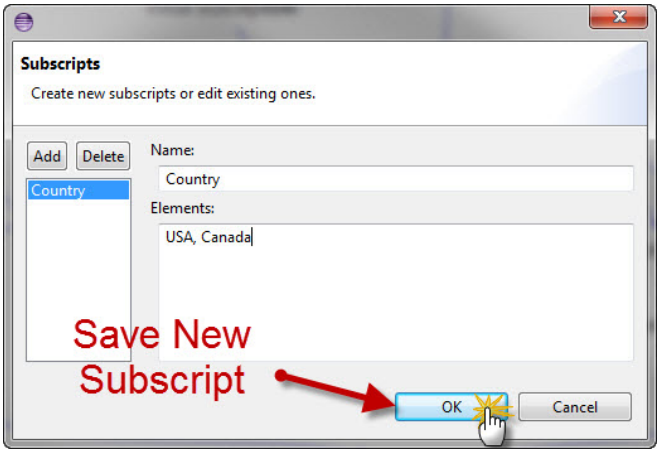

FIGURE 52

Note that the existence of arrays in an SD model is not detectable from simply looking at the model diagram as shown in Figure 57. You would need to examine the properties for the variables to determine which are subscripted.

FIGURE 53

FIGURE 54

FIGURE 55

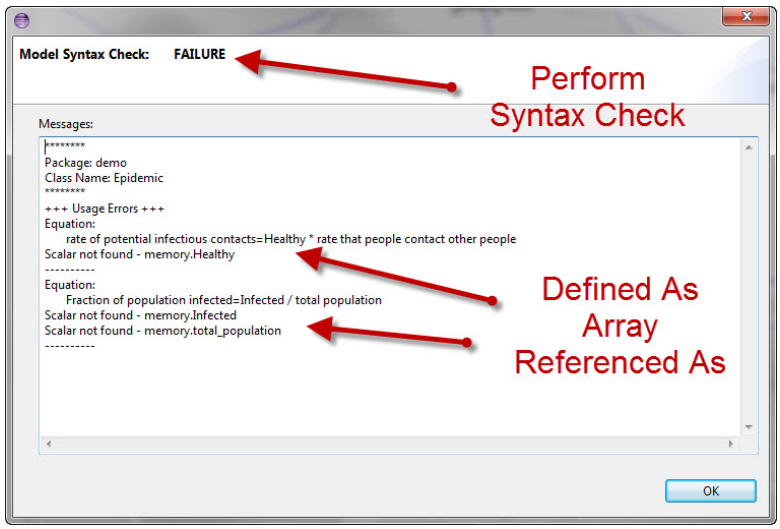

FIGURE 56

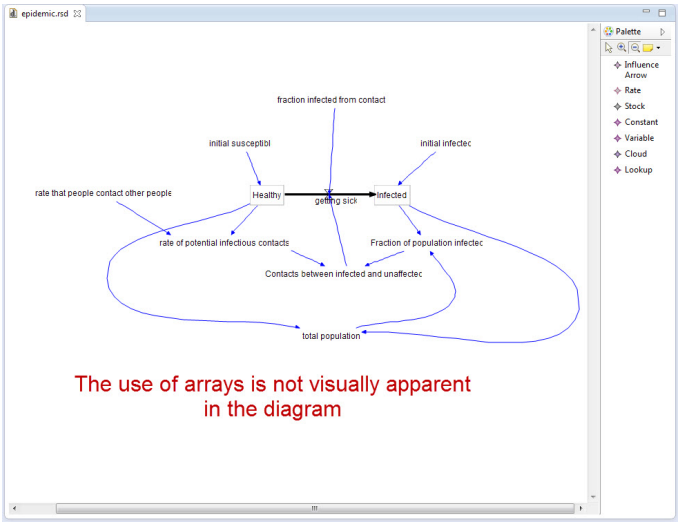

FIGURE 57